cifar_image_prediction

February 20, 2022

1 BUILDING A DENSED MODEL USING KERAS TO PREDICT IMAGES OF OBJECTS USING THE CIFAR DATASET

1.1 loading the Necessary Libraries

[1]:
```
# Libraries for data manipulation, visualization and loading the dataset into␣
↪python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn
import os
from os import path
%matplotlib inline
```

[2]:
```
# Libraries for building our model
from tensorflow.keras.utils import image_dataset_from_directory
from tensorflow.keras.layers import Dense
from tensorflow.keras import layers
from tensorflow import keras
```

1.2 Loading the cifar dataset into python as train_ds, val_ds and test_ds

[3]:
```
file_dir = r'C:\Users\CARNOT\cifar\cifar10\train\airplane'
file_path = file_dir + os.path.sep + '189_airplane.png'
```

[4]:
```
train_ds = image_dataset_from_directory(r'C:\Users\CARNOT\cifar\cifar10\train',␣
↪image_size=(32, 32), seed=612, validation_split=0.3, subset='training')
```

```
Found 50000 files belonging to 10 classes.
Using 35000 files for training.
```

[5]:
```
val_ds = image_dataset_from_directory(r'C:\Users\CARNOT\cifar\cifar10\train',␣
↪image_size=(32, 32), seed=612, validation_split=0.3, subset='validation')
```

```
Found 50000 files belonging to 10 classes.
Using 15000 files for validation.
```

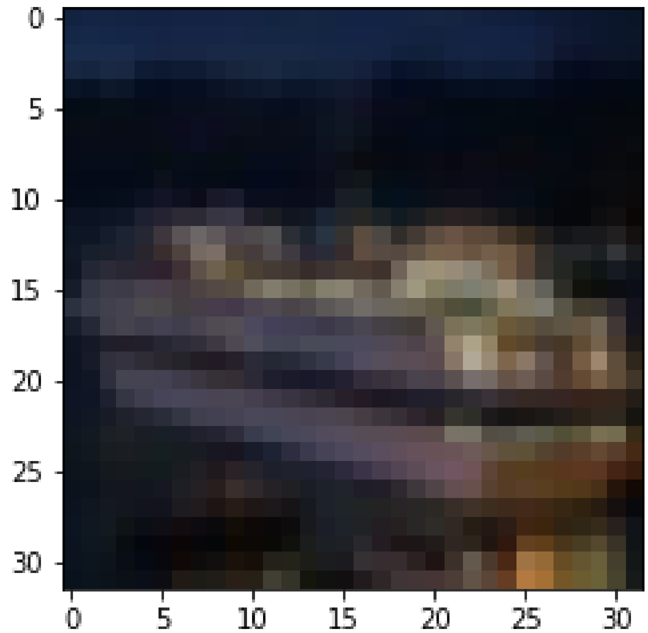
[6]:
```
test_ds = image_dataset_from_directory(r'C:\Users\CARNOT\cifar\cifar10\test',␣
↪image_size=(32, 32), seed=612)
```

```
Found 10000 files belonging to 10 classes.
```

1.3 Preview the image in the train_ds and the label everytime the for loop is run a different image is being generated

[7]:
```
for image, label in train_ds:
    y = label[0]
    plt.imshow(image[0]/255)
    print('label:', label[0])
    break
```

```
label: tf.Tensor(8, shape=(), dtype=int32)
```



1.4 Loading the class_label into python

[8]:
```
class_label = np.array(open(r'C:\Users\CARNOT\cifar\cifar10\labels.txt').read().
↪splitlines())
```

[9]:
```
class_label[5]
```

[9]:
```
'dog'
```

1.5 The model makes use of a sequential model by stacking different layers linearly

- The input into the model which was in the range of 0 - 255 was normalize using the **Rescaling** layer
- The input shape is 3-dimensional in order to use the this input with a dense layer the input is flatten using the **Flatten** layer
- Three hidden dense layer with a neuron of 300, 300 and 100 respectively and an activation function of **relu**
- The output was generated as the last layer with 10 neuron and a softmax activation function

[12]:
```
model = keras.Sequential()
model.add(layers.Rescaling(1./255, input_shape = (32,32,3)))
model.add(layers.Flatten())
model.add(Dense(300, activation='relu'))
model.add(Dense(300, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

1.6 Model summary

[13]:
```
model.summary()
```

```
Model: "sequential_1"

_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_1 (Rescaling)     (None, 32, 32, 3)         0

 flatten_1 (Flatten)         (None, 3072)              0

 dense_4 (Dense)             (None, 300)               921900

 dense_5 (Dense)             (None, 300)               90300

 dense_6 (Dense)             (None, 100)               30100

 dense_7 (Dense)             (None, 10)                1010

=================================================================
Total params: 1,043,310
Trainable params: 1,043,310
Non-trainable params: 0
_________________________________________________________________
```

1.7 Model compilation using Adam as optimizers, sparse_categorical_crossentropy as loss, accuracy as metrics, the model was training using 40 epochs and each epoch as 1094 runs

[28]:
```
model.compile(optimizer=keras.optimizers.Adam(), loss=␣
↪'sparse_categorical_crossentropy', metrics=['accuracy'])
```

[29]:
```
history = model.fit(train_ds, epochs=40, validation_data=val_ds)
```

```
Epoch 1/40
1094/1094 [==============================] - 566s 513ms/step - loss: 1.9042 -
accuracy: 0.3073 - val_loss: 1.7741 - val_accuracy: 0.3535
Epoch 2/40
1094/1094 [==============================] - 795s 721ms/step - loss: 1.7202 -
accuracy: 0.3855 - val_loss: 1.7156 - val_accuracy: 0.3787
Epoch 3/40
1094/1094 [==============================] - 579s 527ms/step - loss: 1.6354 -
accuracy: 0.4105 - val_loss: 1.6073 - val_accuracy: 0.4239
Epoch 4/40
1094/1094 [==============================] - 582s 529ms/step - loss: 1.5824 -
accuracy: 0.4331 - val_loss: 1.6667 - val_accuracy: 0.4041
Epoch 5/40
1094/1094 [==============================] - 666s 606ms/step - loss: 1.5444 -
accuracy: 0.4437 - val_loss: 1.5789 - val_accuracy: 0.4320
Epoch 6/40
1094/1094 [==============================] - 679s 615ms/step - loss: 1.5030 -
accuracy: 0.4591 - val_loss: 1.5532 - val_accuracy: 0.4508
Epoch 7/40
1094/1094 [==============================] - 556s 506ms/step - loss: 1.4747 -
accuracy: 0.4724 - val_loss: 1.5736 - val_accuracy: 0.4382
Epoch 8/40
1094/1094 [==============================] - 556s 505ms/step - loss: 1.4431 -
accuracy: 0.4848 - val_loss: 1.5903 - val_accuracy: 0.4352
Epoch 9/40
1094/1094 [==============================] - 431s 391ms/step - loss: 1.4205 -
accuracy: 0.4876 - val_loss: 1.5506 - val_accuracy: 0.4548
Epoch 10/40
1094/1094 [==============================] - 64s 58ms/step - loss: 1.3995 -
accuracy: 0.4967 - val_loss: 1.5652 - val_accuracy: 0.4454
Epoch 11/40
1094/1094 [==============================] - 65s 59ms/step - loss: 1.3694 -
accuracy: 0.5090 - val_loss: 1.5495 - val_accuracy: 0.4603
Epoch 12/40
1094/1094 [==============================] - 63s 57ms/step - loss: 1.3509 -
accuracy: 0.5134 - val_loss: 1.5811 - val_accuracy: 0.4575
Epoch 13/40
1094/1094 [==============================] - 140s 128ms/step - loss: 1.3254 -
accuracy: 0.5221 - val_loss: 1.5500 - val_accuracy: 0.4638
```

```
Epoch 14/40
1094/1094 [==============================] - 44375s 41s/step - loss: 1.3122 -
accuracy: 0.5302 - val_loss: 1.5758 - val_accuracy: 0.4556
Epoch 15/40
1094/1094 [==============================] - 416s 377ms/step - loss: 1.2887 -
accuracy: 0.5337 - val_loss: 1.5457 - val_accuracy: 0.4737
Epoch 16/40
1094/1094 [==============================] - 56s 51ms/step - loss: 1.2663 -
accuracy: 0.5431 - val_loss: 1.5818 - val_accuracy: 0.4659
Epoch 17/40
1094/1094 [==============================] - 57s 52ms/step - loss: 1.2496 -
accuracy: 0.5496 - val_loss: 1.6256 - val_accuracy: 0.4568
Epoch 18/40
1094/1094 [==============================] - 58s 53ms/step - loss: 1.2406 -
accuracy: 0.5558 - val_loss: 1.6701 - val_accuracy: 0.4480
Epoch 19/40
1094/1094 [==============================] - 180s 165ms/step - loss: 1.2216 -
accuracy: 0.5590 - val_loss: 1.6802 - val_accuracy: 0.4444
Epoch 20/40
1094/1094 [==============================] - 403s 368ms/step - loss: 1.2071 -
accuracy: 0.5632 - val_loss: 1.6595 - val_accuracy: 0.4576
Epoch 21/40
1094/1094 [==============================] - 47s 43ms/step - loss: 1.1880 -
accuracy: 0.5719 - val_loss: 1.7289 - val_accuracy: 0.4506
Epoch 22/40
1094/1094 [==============================] - 48s 44ms/step - loss: 1.1769 -
accuracy: 0.5747 - val_loss: 1.7229 - val_accuracy: 0.4551
Epoch 23/40
1094/1094 [==============================] - 94s 86ms/step - loss: 1.1569 -
accuracy: 0.5830 - val_loss: 1.7427 - val_accuracy: 0.4511
Epoch 24/40
1094/1094 [==============================] - 48s 44ms/step - loss: 1.1443 -
accuracy: 0.5869 - val_loss: 1.7318 - val_accuracy: 0.4586
Epoch 25/40
1094/1094 [==============================] - 48s 44ms/step - loss: 1.1325 -
accuracy: 0.5899 - val_loss: 1.7409 - val_accuracy: 0.4603
Epoch 26/40
1094/1094 [==============================] - 49s 44ms/step - loss: 1.1182 -
accuracy: 0.5953 - val_loss: 1.7800 - val_accuracy: 0.4569
Epoch 27/40
1094/1094 [==============================] - 206s 188ms/step - loss: 1.0962 -
accuracy: 0.6010 - val_loss: 1.8281 - val_accuracy: 0.4409
Epoch 28/40
1094/1094 [==============================] - 46s 42ms/step - loss: 1.0931 -
accuracy: 0.6047 - val_loss: 1.8418 - val_accuracy: 0.4486
Epoch 29/40
1094/1094 [==============================] - 48s 44ms/step - loss: 1.0814 -
accuracy: 0.6072 - val_loss: 1.8777 - val_accuracy: 0.4429
```

```
Epoch 30/40
1094/1094 [==============================] - 48s 43ms/step - loss: 1.0558 -
accuracy: 0.6217 - val_loss: 1.8269 - val_accuracy: 0.4593
Epoch 31/40
1094/1094 [==============================] - 48s 44ms/step - loss: 1.0513 -
accuracy: 0.6207 - val_loss: 1.8703 - val_accuracy: 0.4557
Epoch 32/40
1094/1094 [==============================] - 193s 177ms/step - loss: 1.0454 -
accuracy: 0.6230 - val_loss: 1.8665 - val_accuracy: 0.4562
Epoch 33/40
1094/1094 [==============================] - 59s 54ms/step - loss: 1.0350 -
accuracy: 0.6249 - val_loss: 1.9442 - val_accuracy: 0.4533
Epoch 34/40
1094/1094 [==============================] - 223s 203ms/step - loss: 1.0237 -
accuracy: 0.6271 - val_loss: 1.9200 - val_accuracy: 0.4509
Epoch 35/40
1094/1094 [==============================] - 56s 51ms/step - loss: 1.0142 -
accuracy: 0.6315 - val_loss: 2.0447 - val_accuracy: 0.4416
Epoch 36/40
1094/1094 [==============================] - 57s 52ms/step - loss: 0.9980 -
accuracy: 0.6359 - val_loss: 1.9623 - val_accuracy: 0.4515
Epoch 37/40
1094/1094 [==============================] - 57s 52ms/step - loss: 0.9830 -
accuracy: 0.6428 - val_loss: 1.9412 - val_accuracy: 0.4529
Epoch 38/40
1094/1094 [==============================] - 60s 55ms/step - loss: 0.9798 -
accuracy: 0.6457 - val_loss: 2.0801 - val_accuracy: 0.4519
Epoch 39/40
1094/1094 [==============================] - 57s 52ms/step - loss: 0.9711 -
accuracy: 0.6462 - val_loss: 2.1421 - val_accuracy: 0.4499
Epoch 40/40
1094/1094 [==============================] - 189s 172ms/step - loss: 0.9544 -
accuracy: 0.6537 - val_loss: 2.0978 - val_accuracy: 0.4437
```

[30]:
```
history.params
```

[30]:
```
{'verbose': 1, 'epochs': 40, 'steps': 1094}
```

[31]:
```
history.history
```

[31]:
```
{'loss': [1.9041796922683716,
  1.7201708555221558,
  1.6354111433029175,
  1.582448959350586,
  1.5444411039352417,
  1.5029915571212769,
  1.4747072458267212,
  1.4431449174880981,
```

```
  1.4204879999160767,
  1.399451494216919,
  1.369408130645752,
  1.3508589267730713,
  1.3253555297851562,
  1.3121525049209595,
  1.2886923551559448,
  1.266309380531311,
  1.249582052230835,
  1.2405872344970703,
  1.221564531326294,
  1.2071162462234497,
  1.1879570484161377,
  1.176931619644165,
  1.1569026708602905,
  1.1442818641662598,
  1.1325427293777466,
  1.1181720495224,
  1.0962464809417725,
  1.093075156211853,
  1.081420660018921,
  1.0558325052261353,
  1.0513145923614502,
  1.0453780889511108,
  1.0349661111831665,
  1.0236907005310059,
  1.0142441987991333,
  0.9980206489562988,
  0.9829750061035156,
  0.9798172116279602,
  0.9711115956306458,
  0.9543758034706116],
 'accuracy': [0.30731427669525146,
  0.3855142891407013,
  0.41048571467399597,
  0.43308570981025696,
  0.44371429085731506,
  0.45905715227127075,
  0.4724000096321106,
  0.48482856154441833,
  0.4876285791397095,
  0.496742844581604,
  0.5089714527130127,
  0.5134285688400269,
  0.5220857262611389,
  0.5302000045776367,
  0.5337142944335938,
```

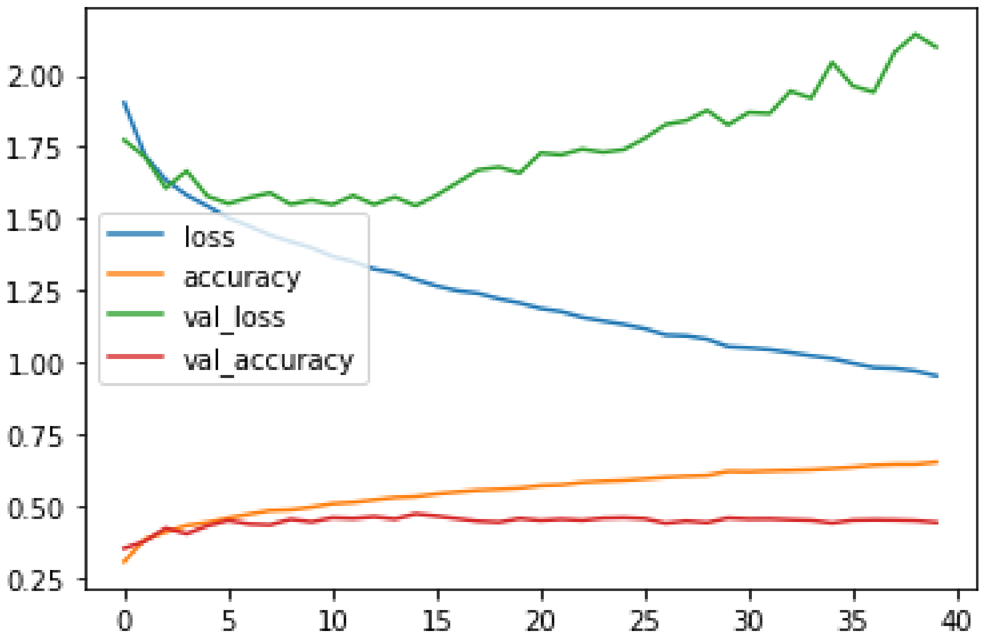
```
 0.5430856943130493,
 0.5496000051498413,
 0.5557714104652405,
 0.5590285658836365,
 0.5631714463233948,
 0.5719143152236938,
 0.5747143030166626,
 0.5830000042915344,
 0.5869428515434265,
 0.5898857116699219,
 0.5953428745269775,
 0.6010000109672546,
 0.6046857237815857,
 0.607200026512146,
 0.621657133102417,
 0.6206571459770203,
 0.623028576374054,
 0.6249428391456604,
 0.6271428465843201,
 0.6314857006072998,
 0.6359142661094666,
 0.6427714228630066,
 0.6456571221351624,
 0.6462000012397766,
 0.6537142992019653],
'val_loss': [1.774119257926941,
 1.715562105178833,
 1.6072523593902588,
 1.6666579246520996,
 1.5788564682006836,
 1.5532466173171997,
 1.5736186504364014,
 1.5903078317642212,
 1.5506318807601929,
 1.5652120113372803,
 1.5495120286941528,
 1.5810949802398682,
 1.5499563217163086,
 1.5758280754089355,
 1.5457062721252441,
 1.5818170309066772,
 1.6255639791488647,
 1.6701351404190063,
 1.680235505104065,
 1.659532904624939,
 1.7289433479309082,
 1.7229282855987549,
```

```
 1.7427326440811157,
 1.7317595481872559,
 1.7408643960952759,
 1.7799824476242065,
 1.8281340599060059,
 1.8418056964874268,
 1.8777207136154175,
 1.8268696069717407,
 1.8702847957611084,
 1.8664730787277222,
 1.94423246383667,
 1.9200104475021362,
 2.044743061065674,
 1.9622842073440552,
 1.9411739110946655,
 2.0801432132720947,
 2.1421267986297607,
 2.0977718830108643],
'val_accuracy': [0.353466659784317,
 0.3787333369255066,
 0.4238666594028473,
 0.40413331985473633,
 0.4320000112056732,
 0.45080000162124634,
 0.4381999969482422,
 0.4352000057697296,
 0.454800009727478,
 0.4453999996185303,
 0.46026667952537537,
 0.45746666193008423,
 0.46380001306533813,
 0.45559999346733093,
 0.4736666679382324,
 0.46586665511131287,
 0.45680001378059387,
 0.4480000138282776,
 0.44440001249313354,
 0.4575999975204468,
 0.4505999982357025,
 0.4550666809082031,
 0.45106667280197144,
 0.4586000144481659,
 0.46033334732055664,
 0.45686665177345276,
 0.4408666789531708,
 0.44859999418258667,
 0.4429333209991455,
```

```
    0.45926666259765625,
    0.4557333290576935,
    0.4562000036239624,
    0.4533333480358124,
    0.4509333372116089,
    0.4415999948978424,
    0.451466795730591,
    0.45286667346954346,
    0.4519333243370056,
    0.4498666524887085,
    0.4436666667461395]}
```

[34]:
```
# Checking the performance of our model
pd.DataFrame(history.history).plot()
```

[34]: <AxesSubplot:>



[35]:
```
# evaluating the model on the test data
model.evaluate(test_ds)
```

```
313/313 [==============================] - 122s 371ms/step - loss: 2.0311 -
accuracy: 0.4485
```

[35]: [2.0310769081115723, 0.44850000739097595]

[43]:
```python
# generation some input from the test_ds to be used in predicting
for img, lb in test_ds:
    print(img[:4].shape)
    print(lb[:4].shape)
    image = img[:4]
    label = lb[:4]
    break
```

```
(4, 32, 32, 3)
(4,)
```

[45]:
```python
# running a prediction on the model
prob_predict = model.predict(image)
```

[48]:
```python
# predicted label
predict = class_label[prob_predict.argmax(axis = 1)]
predict
```

[48]:
```
array(['dog', 'deer', 'dog', 'truck'], dtype='<U10')
```

[53]:
```python
# actual label
actual_label = class_label[np.array(label)]
actual_label
```

[53]:
```
array(['dog', 'deer', 'horse', 'dog'], dtype='<U10')
```

[55]:
```python
# weights and biases of the first dense layer
model.layers[2].get_weights()
```

[55]:
```
[array([[-2.60950904e-02, -2.63698753e-02,  2.68406868e-02, …,
          1.22417927e-01, -4.44985405e-02, -1.13394083e-02],
        [-3.13659497e-02, -1.86714586e-02, -3.52070071e-02, …,
          1.20468535e-01,  6.07513539e-05,  9.65113472e-03],
        [-8.15506000e-03, -3.80092375e-02, -3.07999784e-03, …,
          6.27948999e-01,  2.08293460e-02,  1.99701581e-02],
        …,
        [ 5.96103352e-03, -3.96754332e-02, -4.66282107e-03, …,
         -1.50388092e-01, -1.28507577e-02, -3.39771360e-02],
        [-6.57599093e-03, -3.77745926e-02, -3.02913096e-02, …,
         -2.41550878e-01,  3.38570513e-02,  1.34094637e-02],
        [-1.80551987e-02,  3.60735916e-02, -6.63135340e-03, …,
         -3.27629596e-01, -2.00156402e-02, -3.31095420e-02]], dtype=float32),
 array([-2.7262277e-03, -5.9995563e-03, -2.9328512e-03, -5.9883334e-03,
        -4.4228504e-03,  1.5413033e+00, -4.2088493e-03, -6.6208933e-03,
        -2.1069276e-03, -5.9963623e-03, -6.0012247e-03, -6.2797377e-03,
        -6.0036923e-03, -6.0037398e-03, -6.0011339e-03,  4.4436587e-04,
        -3.1507770e-03, -9.0783918e-03, -6.0045770e-03, -6.0025710e-03,
         7.8846957e-04,  2.0184676e-01, -3.8337836e-03, -3.9372887e-03,
```

```
-5.9987493e-03, -3.5372996e-03, -4.1434430e-03, -5.9956238e-03,
-5.3759189e-03, -6.0032187e-03, -3.4120954e-03, -6.3189804e-03,
-8.2087945e-03, -6.0046571e-03, -6.8694712e-03, -1.8062337e-03,
-3.3497144e-03, -5.0610043e-03, -6.0015027e-03, -4.1776351e-03,
-5.3815977e-03,  3.8771930e-01, -2.1732152e-03, -2.8323510e-03,
-2.3545537e-03, -3.2312986e-03, -5.4151677e-03, -4.3712556e-03,
-5.0980682e-03, -5.2915180e-01, -1.7915209e-03, -5.1077618e-03,
-6.0021444e-03, -5.9701446e-03, -5.5939786e-04, -4.5607844e-03,
-1.7191811e-02,  1.8665029e-02, -5.0480645e-03,  1.6871852e-01,
-6.0022129e-03, -6.0034390e-03, -5.9954007e-03, -6.0039531e-03,
-4.3533915e-03, -2.9024445e-03, -2.6402411e-03,  0.0000000e+00,
-2.6887471e-02, -5.1427423e-03, -2.9538956e-03, -5.9991297e-03,
-5.9984582e-03, -4.7602309e-03, -4.7155167e-03, -6.0001020e-03,
-7.9907589e-03, -6.0023759e-03, -4.8191906e-03,  5.7814284e-03,
-5.1726270e-03, -7.0737083e-03, -4.2219697e-03, -2.1395106e-03,
-4.0321900e-03, -4.0402156e-03, -1.1914301e-02, -6.0029523e-03,
-5.7009566e-03, -4.9491873e-04, -2.2253266e-03, -5.2036163e-03,
-8.1571089e-03, -1.0742945e-02, -2.8903414e-03, -6.0017128e-03,
-6.0000303e-03, -5.3288159e-03, -1.3779321e-02, -5.4106326e-03,
-5.9772395e-03, -5.9960717e-03,  6.1196613e-01, -6.4848256e-03,
-4.7605424e-03, -3.8859691e-03, -2.7408681e-03, -5.8352379e-03,
-7.6069781e-03, -5.4679713e-03, -2.4658296e-01, -2.5251940e-02,
 7.5564951e-02,  1.8573581e-01,  8.3781583e-03, -6.3327667e-03,
-6.0019903e-03,  3.2494836e-03, -6.7125862e-03, -2.8718780e-03,
-4.1104913e-01, -9.4327591e-03, -2.9302645e-03, -7.5945631e-03,
-5.9404909e-03, -5.0344891e-03, -4.9969899e-03, -1.1175504e-02,
 2.5874564e-01, -5.9991945e-03, -1.4952232e-02, -6.8771630e-03,
-5.7325605e-03, -1.4713787e-03, -5.9967749e-03, -5.0152298e-03,
-4.5410790e-03, -3.1262696e-01, -5.0859572e-03, -5.9969597e-03,
-3.9751404e-03, -6.0037151e-03, -2.2926329e-02, -5.9945523e-03,
-5.5503477e-03,  6.8447506e-04, -5.9932834e-03, -5.9928098e-03,
-5.9958855e-03, -3.6615266e-03, -3.4499492e-03, -6.0041747e-03,
 1.9155674e-01, -6.0016043e-03, -5.9675346e-03, -3.7998268e-03,
 1.2548244e-01, -5.1068664e-03, -7.0707965e-01, -7.7800211e-03,
-6.0003721e-03, -4.9557486e-03, -5.1241419e-03, -3.8776882e-03,
-3.3879532e-03, -5.9670070e-03, -2.2373738e-02, -3.9272364e-03,
-6.5114810e-03, -4.5132628e-03,  5.4713037e-02, -5.5531119e-03,
-1.5932465e-02, -3.3924181e-03, -3.9654588e-03, -7.5583839e-01,
-6.0045882e-03, -5.5348156e-03, -5.9951032e-03, -2.2214954e-03,
-7.4551860e-04, -6.0040313e-03, -3.0720232e-03,  8.1307134e-03,
 1.5429512e-05,  2.3759100e-01, -1.5578045e-03, -5.2008848e-03,
-6.0043186e-03, -4.7808187e-03, -7.9964073e-03, -6.0036178e-03,
-5.5453763e-03, -1.2982816e-03,  5.6802922e-01, -6.0128290e-03,
-1.4612292e-03, -1.0719941e-03, -1.8480857e-03, -2.8997293e-02,
-9.7281514e-03, -6.0044173e-03, -4.5403009e-03, -7.1715719e-01,
 7.5737327e-01, -8.2222062e-01, -5.9982063e-03, -6.0044178e-03,
-3.0931970e-03, -6.2093320e-03, -6.0001593e-03, -2.5316284e-03,
```

```
   -4.6616388e-03, -7.7979788e-03, -3.1608089e-03, -5.9890808e-03,
   -5.9998273e-03,  2.8340505e-03, -6.6937459e-01, -2.6356688e-02,
   -6.0037761e-03, -5.7530841e-03,  7.1887147e-01, -6.0040746e-03,
   -1.1510536e-02, -2.8279242e-03, -6.0012904e-03, -2.6433924e-03,
   -5.6543476e-03, -6.6278912e-03, -2.9161149e-01, -3.9945659e-03,
   -4.7552547e-01, -5.9784553e-03, -3.7320933e-01, -5.1522572e-03,
   -3.6926535e-03, -6.0039256e-03, -5.3027510e-03, -8.1782201e-03,
   -6.0017570e-03, -3.5224222e-03, -6.7469985e-03, -6.0000396e-03,
   -4.4581308e-03, -8.4061110e-03, -4.2614895e-03, -9.6494652e-04,
   -2.7737562e-03, -6.6959448e-03, -4.8928251e-03, -2.9277746e-02,
   -2.6606945e-03, -2.5910456e-03, -2.2524269e-03, -2.3441380e-02,
   -3.0234221e-03, -6.0005188e-03, -4.3026009e-03, -3.2774091e-03,
   -6.9785952e-03, -5.9986282e-03, -6.0030320e-03,  4.5211205e-01,
   -6.0029249e-03, -4.6370071e-03, -5.1199007e-03,  9.9542970e-03,
    3.4318953e-03, -6.0015237e-03, -3.7805848e-02, -5.9986338e-03,
   -3.6820897e-04, -5.5380366e-03, -3.6893634e-03,  2.1003947e-01,
   -6.0013854e-03, -5.3271633e-03, -2.1145991e-03,  3.6396897e-01,
   -6.0032210e-03, -6.0017798e-03, -6.0038995e-03, -5.9876652e-03,
   -5.2631279e-03, -5.5277525e-03, -6.2378407e-03, -6.0015619e-03,
   -2.2467996e-03,  3.1873517e-02, -6.0047046e-03, -6.0008084e-03,
   -5.9959358e-03, -4.1677649e-03, -9.2731006e-03, -8.6660990e-03,
   -6.8326038e-03, -6.1926931e-01, -6.0004154e-03, -5.9973132e-03],
  dtype=float32)]
```

[56]:
```
# Saving my model
model.save('cifar.hf5')
```

```
INFO:tensorflow:Assets written to: cifar.hf5\assets
```